Disciplina : Programação Orientada a Objetos - 2018/1
Professora : Nádia Félix Felipe da Silva
Prova 2

Data : 15/08/2022

Aluno(a) : _____

1. Responda V ou F (2,0):

( ) Classes Abstratas e Interface não podem ser usadas para instanciar objetos mas podem ser usadas com o objetivo de reaproveitar código através de herança.

( ) Interface é uma forma de implementar herança múltipla em Java.

( ) Polimorfismo é um conceito relacionado a métodos de mesma assinatura terem implementações diferentes. Ele só é possível quando existe um relacionamento de herança entre as classes.

( ) Exceção é um evento que interrompe o fluxo normal de um sistema. O java obriga o tratamento de exceções de algumas operações, e outras, não obrigatórias, podem ser tratadas para que o sistema não apresente o erro para o usuário. Elas são chamadas respectivamente de *unchecked* e *checked.*

( ) Duas classes podem estar relacionadas através de um relacionamento todo-parte de agregação ou de composição. No primeiro, a parte é dependente do todo e no segundo não.

( ) A criação de Classes Abstratas e interfaces é uma forma de organizar os dados generalizando operações e características comuns de várias classes concretas.

( ) Os diagramas de casos de uso e classes são importantes na documentação de um sistema. O diagrama de classes é o principal diagrama para levantamento de requisitos com os usuários.

( ) Utilizando herança podemos fazer com que um mesmo objeto se comporte de formas diferentes na aplicação. Esta ideia se refere aos conceitos de abstração e encapsulamento.

2. Dadas as classes Pessoa, Dívida e Principal, parcialmente implementadas. Considerando que uma pessoa pode ter vários tipos de Dívidas em seu nome e que a relação entre pessoas e dívida pode ser considerada uma composição. Complete os códigos das classes Pessoa, Divida e Principal para atender aos seguintes requisitos (4,0):
   a) Uma pessoa pode ter várias dívidas – implementar a relação/vínculo entre as classes;
   b) Criar os métodos invocados pela classe Principal:

- **criaDivida** – que recebe os dados de uma dívida, cria um nova dívida com esses dados e adiciona à lista de dívidas de uma pessoa;
- **mostraDívidas** – mostra todos os dados das Dívidas de uma pessoa (usando o método toString da classe Dívida;

   c) Criar os seguintes métodos na classe Dívida:

- **pagaPrestacao** - recebe uma quantidade de prestações pagas e atualiza o atributo qtdePagas acrescentando a ele, a quantidade recebida como parâmetro;
- **calculaPrestacao** – retorna o valor da prestação calculado dividindo o valor total pela quantidade de prestações;
- **calculaSaldoDevedor** – retorna o valor do saldo devedor que é calculado diminuindo do valor total as prestações já pagas;

   d) Instancie uma nova pessoa com 3 tipos de dívidas diferentes sendo que um deles deve ter 36 prestações com 5 já pagas, e um outro deve ter 10 prestações com 8 já pagas.

```
import java.util.ArrayList;
public class Pessoa {
        private String nome;
        private String cpf;
        Pessoa (String nome, String cpf) {
                this.nome = nome;
                this.cpf = cpf;
                }
```

```
public class Principal {
        public static void main(String[] args) {
                Pessoa p = new Pessoa("Ana","123456");
                p.criaDivida("Pessoal", 10000.00, 20);
                p.criaDivida("Veículo", 20000.00, 30);
                System.out.println(p);
                p.mostraDividas();
        }
```

Disciplina : Programação Orientada a Objetos - 2018/1
Professora : Nádia Félix Felipe da Silva
Prova 2

Data : 15/08/2022

Aluno(a) : _____

```
        public String toString() {
                return "Nome : " + nome + " - CPF : " + cpf;

        }
}
```

```
}
```

```
public class Divida {
        String descricao;
        double valorTotal;      // valor total da Dívida
        int qtdePrestacoes;     // quantidade total de prestações da dívida
        int qtdePagas;          // quantidade de prestações paga até o momento
        Divida (String descricao, double valorTotal, int qtdePrestacoes) {
                this.descricao = descricao;
                this.valorTotal = valorTotal;
                this.qtdePrestacoes = qtdePrestacoes;
        }
        public String toString() {
                return descricao + " - Valor: " + valorTotal +
                                " - quantidade de prestacoes : " + qtdePrestacoes +
                                " - saldo devedor : " + this.calculaSaldoDevedor() + " - prestacao : " +
                                this.calculaPrestacao();
        }
}
```

3. Considerando as classes da questão anterior, represente-as em UML de forma **completa** (todos os relacionamentos, atributos, métodos, tipos de dados, multiplicidade, navegabilidade). Usando o conceito de herança, modele também duas classes específicas que representam as dívidas imobiliárias e as dívidas estudantis, com algumas características a mais. As imobiliárias têm as as seguintes características diferentes: tipo de imóvel (casa, apartamento, lote) e quantidade de metros; e as estudantis têm a faculdade e o curso (4,0).